Authors: @rcatalan98 and @unbalancedparenthese

tldr

: Bridges are generally insecure and economically inefficient. They exhibit an asymmetry between users and bridge operators, where users can easily lose funds. We propose a bridge design that is simple, modular, and utilizes multi-storage proofs and the native messaging system between Ethereum and Layer 2 networks (L2s) as a fallback mechanism.

Bridging is a trust issue

How can we offer a system where the users don't have to trust a facilitator to exchange their assets from an L2 to Ethereum?

We propose a simple protocol that follows these steps:

1. The user specifies a destination address on Ethereum and locks the tokens X to be bridged into an L2 escrow smart contract.
2. A market maker monitors a change of state in the escrow smart contract.
3. a. The market maker calls the transfer function of the PaymentRegistry Contract in Ethereum.

b. The transfer function of the PaymentRegistry contract in Ethereum pays the tokens X to the User.

1. A storage proof is generated, containing evidence of a transfer from the market maker's Ethereum account to the user-specified address in Ethereum.
2. Ethereum PaymentRegistry storage information is used as part of a storage proof.
3. L2 Escrow contract verifies the storage proof of the PaymentRegistry contract in Ethereum and pays the MM with the initial tokens locked by the user.

[

2346×1428 543 KB

](https://ethresear.ch/uploads/default/original/2X/a/a30bc29313a33e3679ed4c345298b1cd74cdfe35.png)

The same design can be expanded to be used to bridge tokens from an L2 to another L2. The same design can include multi-proof storage proofs instead of using only one. We also have implemented a fallback mechanism using the native message mechanism between Ethereum and L2s in case the storage proof providers are offline.

Fallback mechanism

If the storage proof providers are not available, the market maker can prove to the Escrow contract that they fulfilled the user's intent through the rollup's native messaging system. Using this messaging system has the same trust assumptions as the L2s used in the transfer.

Risks

For the user, the risks include the existence of a bug in the code of the smart contract, the existence of a bug in the circuits of the ZK/validity proof verification and the fact that the storage proof provider can go offline. The first risk is mitigated by having a very simple smart contract. The second risk is mitigated by using multi-proof storage proofs and multiple ZK/validity proof implementations or TEEs. If the storage proof provider goes offline the fallback mechanism can be used.

The risks for market makers are the same as for users, plus the risk of reorganization of the chain and the fact that the market maker receives the same tokens on the L2s rather than on Ethereum.

Since the capital is locked for a short period (until the proof is generated or the message arrives), the risks are minimized and the attack surface is smaller for the market maker.

Questions

What are our disadvantages?

The biggest disadvantage of this solution is that users can only bridge tokens that are present in both the origin and destination chains.

Another disadvantage is that the risks don't disappear; they are simply transferred to the market maker.

How can users cancel their order?

Initially, we are not going to offer the ability to cancel orders. The main reason is to avoid any timing attacks. For instance, a user could create an order and cancel it right after the market maker has paid them on the destination chain, thereby stealing funds from the market maker.

Is there any real-world implementation of this bridge?

Yes. We have already implemented this between Starknet and Ethereum. We plan to integrate zkSync, Arbitrum, Optimism, Scroll, Base, and Linea next.

All integrations require the same codebase with a few modifications, except for Starknet, which is not EVM compatible.

How fast is it?

From the user's perspective, the bridging is completed in less than 30 seconds, as quickly as the time it takes the market maker to observe the user's deposit and execute a transfer.

From the market maker's perspective, they will be able to withdraw the money after paying the user and generating the storage proof. This normally takes between 5 and 15 minutes. It's important to also consider that the market maker will need to rebalance their liquidity using the native bridge and wait for the finality of the native bridge to rebalance their portfolio.

How cheap is it?

The cost of this bridge is similar to an ERC20 transfer plus the cost of proving the state of the L1 and L2. This second cost tends towards zero since it's amortized by multiple calls that use the same proof, and the proving cost is minimal compared to on-chain transfers.

What is new in this design? Didn't storage proofs solve this problem already?

Storage proofs alone don't fundamentally change the design of a traditional bridge. They merely enable a safer coordination mechanism.

Locking the user's capital first provides guarantees to the market maker that they will receive the funds in exchange for fulfilling the user's intent.

Couldn't you solve this problem without Storage Proofs? What do they add to the table?

Yes, Storage Proofs are not 100% necessary to solve this problem. But they are a key technological component for a future proof architecture. If we want this protocol to scale, storage proofs are the best way to do this. It will allow us to prove many orders together.

What are the benefits against an Optimistic Oracle?

Optimistic Oracles were a great solution before Storage Proofs were a feasible solution, their main disadvantages are:

- Optimistic Oracles relay in Game Theory to work and it's difficult to bootstrap an ecosystem to make them robust.
- Codebases are complex
- The settlement period takes a few hours (depending on the solution) and end up creating inefficiencies for the market makers.

On the other hand our protocol with the native messaging and storage proofs takes no longer than 15 minutes (between Ethereum and L2) to unlock the funds. The protocol codebase is no more than 500 lines of code and the risks are easy to understand by all the players and therefore easy to come up with ways to mitigate them.

Did anybody do something similar beforehand?

Hop Protocol was one of the first bridges to allow cross-chain swaps between rollups and Ethereum with an AMM-based design using multiple messaging systems (native and optimistic). The main issue lies in the capital inefficiency of an AMM model and the significant security risks of locking large amounts of capital in complex cross-chain communications.

Across was the first bridge to leverage intents for a faster bridge experience and lower capital costs per transaction. However, by using an Optimistic Oracle, it naturally has a challenging period that the market has to wait to get its funds back. To optimize some of the problems their settlement mechanism introduces, they offer financial products around their main bridge solution, such as Liquidity Pools that front the capital to the market makers.

What are our advantages?

Our bet is that zero-knowledge proofs will continue to improve, becoming faster and safer, thus enhancing our solution and allowing us to offer better prices by lowering risks and shortening the repayment period.

- Fast and cheap bridging experience for the user
- Short capital lock-up period for the market maker
- Low on-chain complexity. The smart contracts in total are not larger than 300 lines of code.

Next steps

- Speeding up message passing with EigenLayer to allow cross-chain swap settlements between L2s. The protocol should have the option to send faster messages between rollups and Ethereum with similar trust assumptions of the native messaging system
- Introducing Partially Filled Orders, offering cheaper but slower transfers with batching
- Intent-based DeFi Pooling
- Unified wallet abstraction across multiple L2s